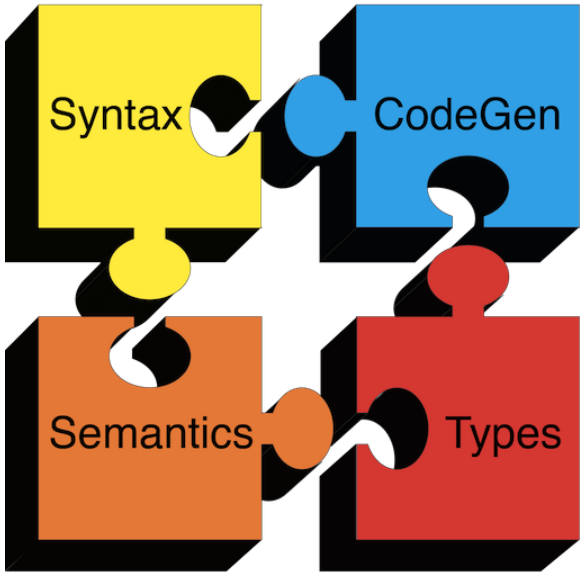


Compilers

Automatic Memory Management

Managing Memory

- Storage management is still a hard problem in modern programming

- C and C++ programs have many storage bugs
  - forgetting to free unused memory
  - dereferencing a dangling pointer
  - overwriting parts of a data structure by accident
  - and so on...

- Storage bugs are hard to find
  - a bug can lead to a visible effect far away in time and program text from the source

Managing Memory

- This is an old problem:
  - studied since the 1950s for LISP
- There are well-known techniques for completely automatic memory management
- Became mainstream with the popularity of Java

Managing Memory

- When an object is created, unused space is automatically allocated
  - In Cool, new objects are created by new X
- After a while there is no more unused space
- Some space is occupied by objects that will never be used again
  - This space can be freed to be reused later

Managing Memory

- How do we know an object will “never be used again”?

- Observation: a program can use only the objects that it can find:

$$\text{let } x : A \leftarrow \text{new } A \text{ in } \{ x \leftarrow y; \ldots \}$$

Managing Memory

- An object x is reachable if and only if:
  - a register contains a pointer to x, or
  - another reachable object y contains a pointer to x
- You can find all reachable objects by starting from registers and following all the pointers
- An unreachable object can never be used
  - such objects are garbage

Managing Memory

- Consider the program:

```
x ← new A;
y ← new B
x ← y;
if alwaysTrue() then x ← new A else x.foo() fi
```

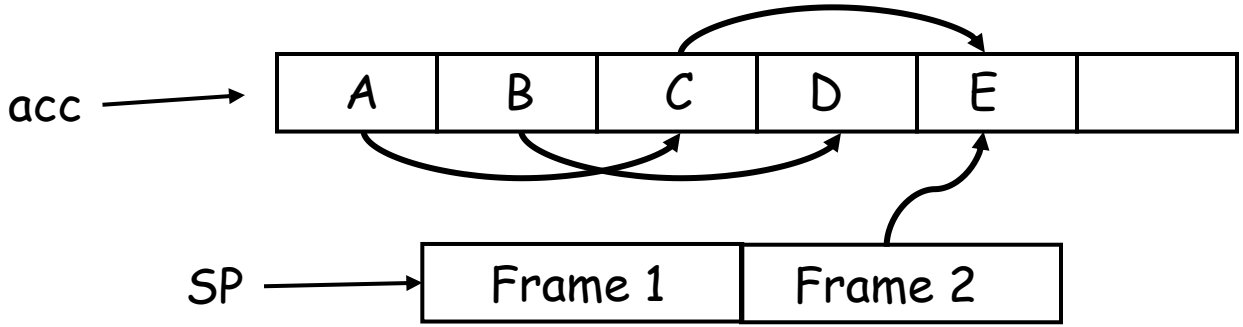
- After x ← y (assuming y becomes dead there)
  - the first object A is unreachable
  - the object B is reachable (through x)
  - thus B is not garbage and is not collected
    - but object B is never going to be used

Managing Memory

- Coolc uses an accumulator
  - it points to an object
  - and this object may point to other objects, etc.

- And a stack pointer
  - each stack frame contains pointers
    - e.g., method parameters
  - each stack frame also contains non-pointers
    - e.g., return address
  - if we know the layout of the frame we can find the pointers in it

Managing Memory



- In coolc we start tracing from acc and stack
  - These are the *roots*
- Note B and D are unreachable from acc and stack
  - Thus we can reuse their storage

Managing Memory

- Every garbage collection scheme has the following steps
  1. Allocate space as needed for new objects
  2. When space runs out:
     a) Compute what objects might be used again (generally by tracing objects reachable from a set of “root” registers)
     b) Free the space used by objects not found in (a)

- Some strategies perform garbage collection before the space actually runs out